

A REAL-TIME PACKET SCAN ARCHITECTURE



Tim Sherwood
UC Santa Barbara

Big Questions

Can my system be optimized further?

- If so, then how and when?
- How much benefit can I expect?
- Have I seen this behavior before?

Is my system working “correctly”?

- Soft errors, backdoors, hardware bugs

Am I under attack?

- If so, then by whom?
- Am I witness to an attack?

Online Monitors

To Protect and Serve

- Our machines are constantly under attack
- Cannot rely on end users, we need networks which *actively defend themselves*.

IDS/IPS are promising ways of providing protection

Market for such systems: \$918.9 million by the end of 2007.

Snort: an widely accepted open source IDS

This requires the protection system to be able to operate at 10 to 40 Gb/s. (We aim at current and next generation networks.)

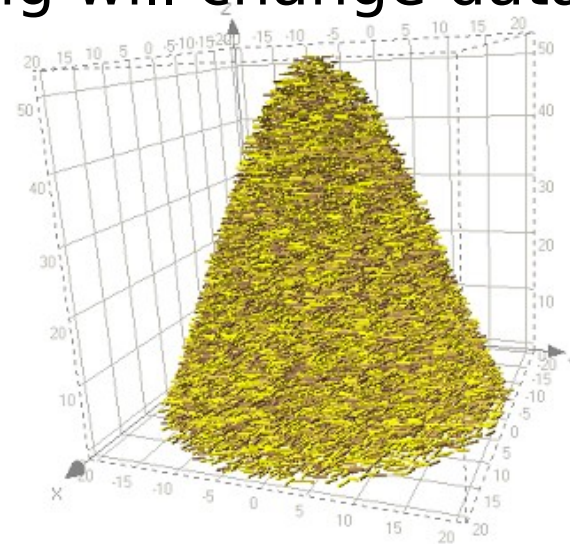
The Problem

Our computing infrastructure is fast

- Processors → $\sim 10^9$ instructions/second
- Network Routers → $\sim 10^9$ bytes/second

Beyond our ability to monitor naively

- Full traces are near impossible to gather
- Sampling may miss important data
- Intrusive monitoring will change data



New Architectures are Required

Why a new Computer Architecture



~~Latency~~

- Throughput is critical
 - 40 Gigabit link = Packet out every 8 ns

~~Common Case~~

- Design for worst case stream
 - Each packet needs multiple memory ref
 - Network vendors chip by wire rate
 - Denial of service and reliability
 - Caches are no help

Packet Scan Architecture

High Performance Packet Scan Architecture

- Underlying primitives to support high-throughput monitors
- Algorithm – Architecture co-design

Example primitive: String Matching

- 0.4MB and 10Gbps for Snort rule set (>10,000 characters)

Bit-Split String Matching Algorithm

- Reduces out edges from 256 to 2.
- Formal language – correctness and efficiency

Memory Tile Based Design

- Memory throughput is the key
- Data is distributed over tiles with bounded contention

Performance/area *beats* the best techniques we examined by a factor of 10 or more.

Packet Scan Architecture

String Matching

*examine
packet content*

Bit-Split String Matching Algorithm

A Memory Tile Based Architecture

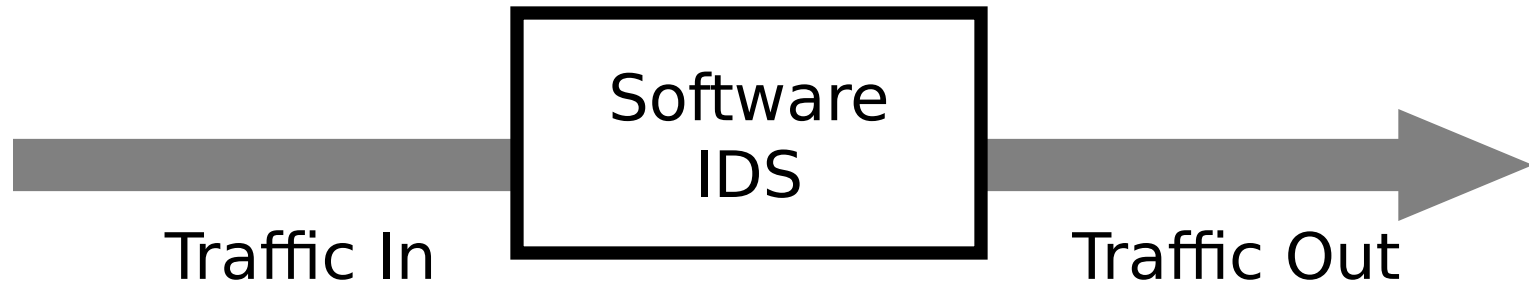
Building a Real System

Is it really correct?

Future Work

Scanning for Intrusions

CodeRed worm:
web flow established
uricontent with
“/root.exe”



Most IDS define a set of
rules.

A *string* defines a suspicious transmission.

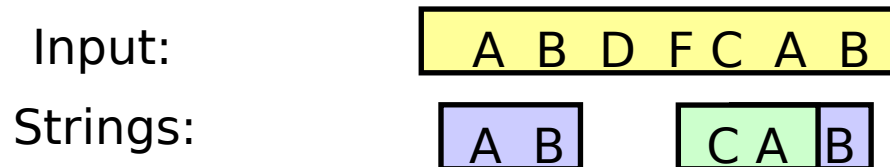
We are not building a full IDS, rather
building the **primitives** from which full
systems can be built

Multiple String Matching

The multiple string matching algorithm:

- Input: A set of strings/patterns ***S***, and a buffer ***b***
- Output: Every occurrence of an element of ***S*** in ***b***

A string can be *anywhere* in the payload of a packet.



- Extra constraint: ***b*** is really a stream

How to implement:

Option 1) search for each string independently

Option 2) combine strings together and search all at once

Why hardware

Snort: >1,000 rules, growing at 1 rule/day or more

Active research into automated rule building

Strings are not limited to be just [a-z]+

We need a high speed string matching technique with *stringent worst case* performance.

Many algorithms are targeted for average case performance. Aho-Corasick can scan once and output all matches. But it is **too big to be on-chip**.

The Aho-Corasick Algorithm

Given a finite set P of patterns, build a deterministic finite automaton G accepting the set of all patterns in P .

The Aho-Corasick Algorithm

An Aho/Corasick String Matching Automaton for a given finite set P of patterns is a (deterministic) finite automaton G accepting the set of all words containing a word of P as a suffix. G consists of the following components:

finite set Q of **states**

finite **alphabet** A

Transition function $g: Q \times A \rightarrow Q + \{\text{fail}\}$

Failure Function $h: Q \rightarrow Q + \{\text{fail}\}$

initial state q_0 in Q

a set F of **final states**

On String Matching and Languages

This should not be any big surprise

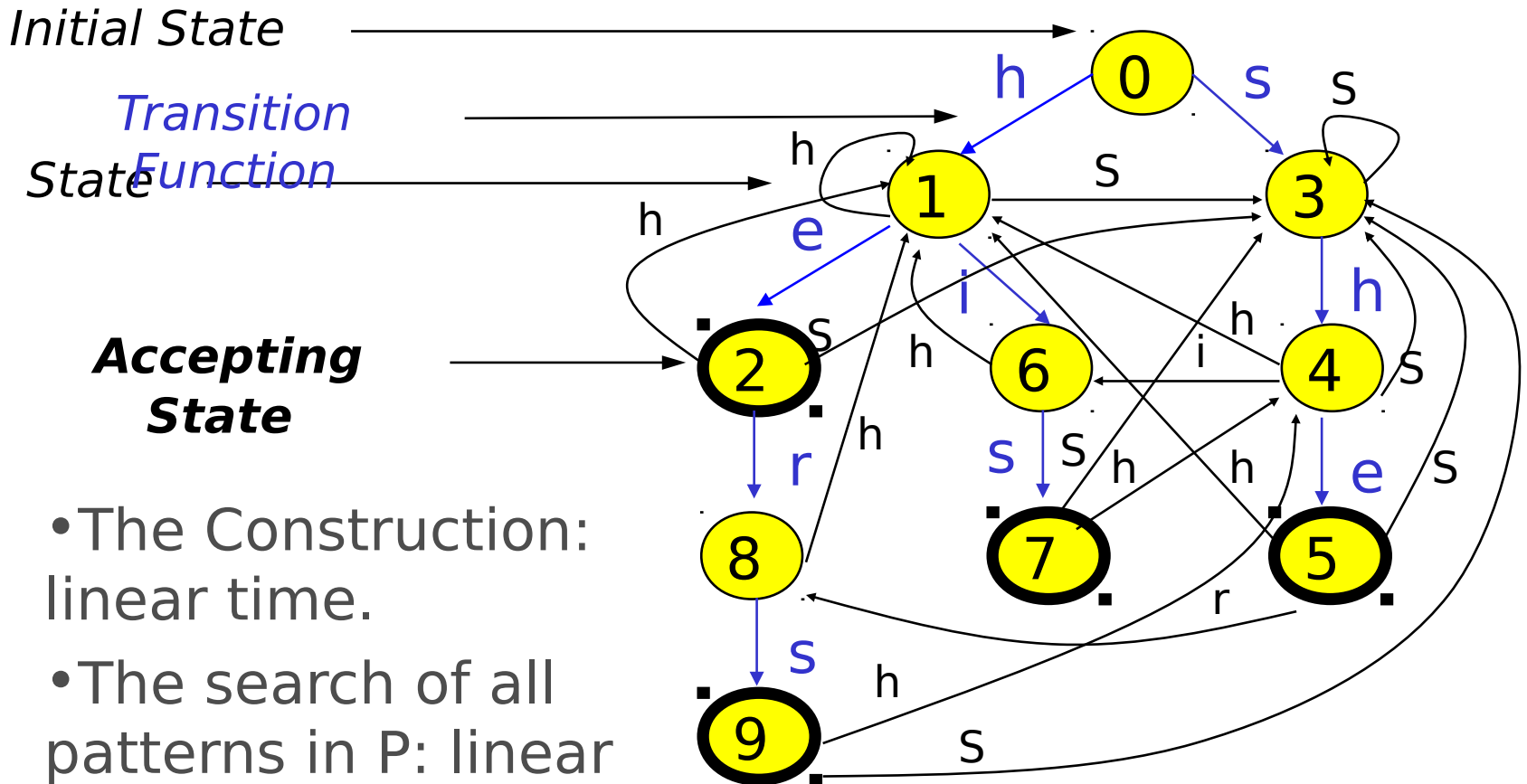
- P is a FL
- $FL \subset RL$
- RL can be recognized by a RE
- RE can be simulated with an NFA
- An NFA can be simulated with a DFA

This last step is the problem

- Aho and Corasick shows that for FL there is no exponential blow up in state

An AC Automaton Example

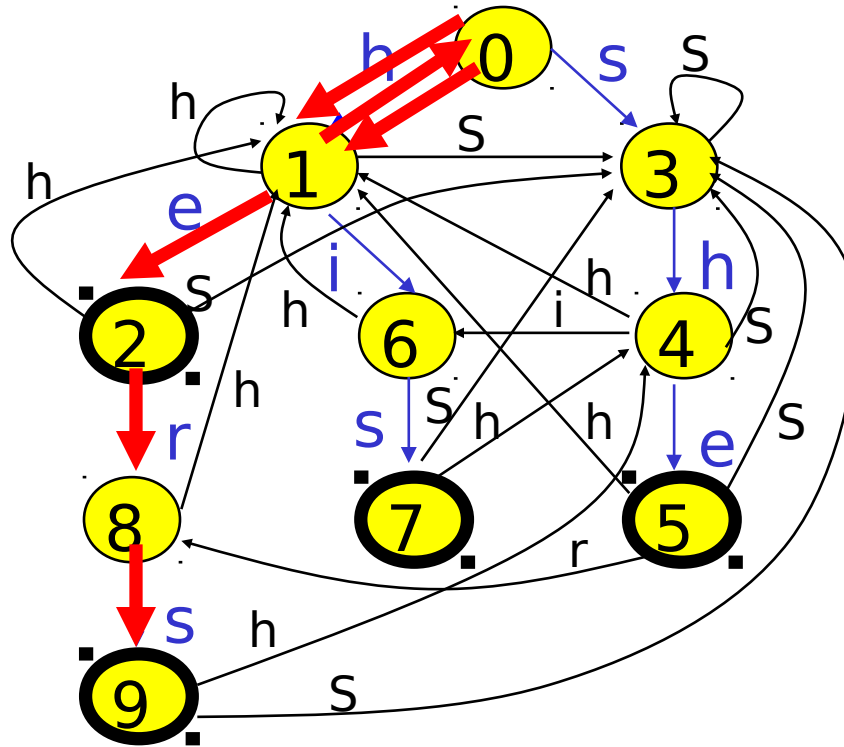
Example: $P = \{\text{he, she, his, hers}\}$



- The Construction: linear time.
- The search of all patterns in P : linear time

(Edges pointing back to State 0 are not shown).

Matching on the example



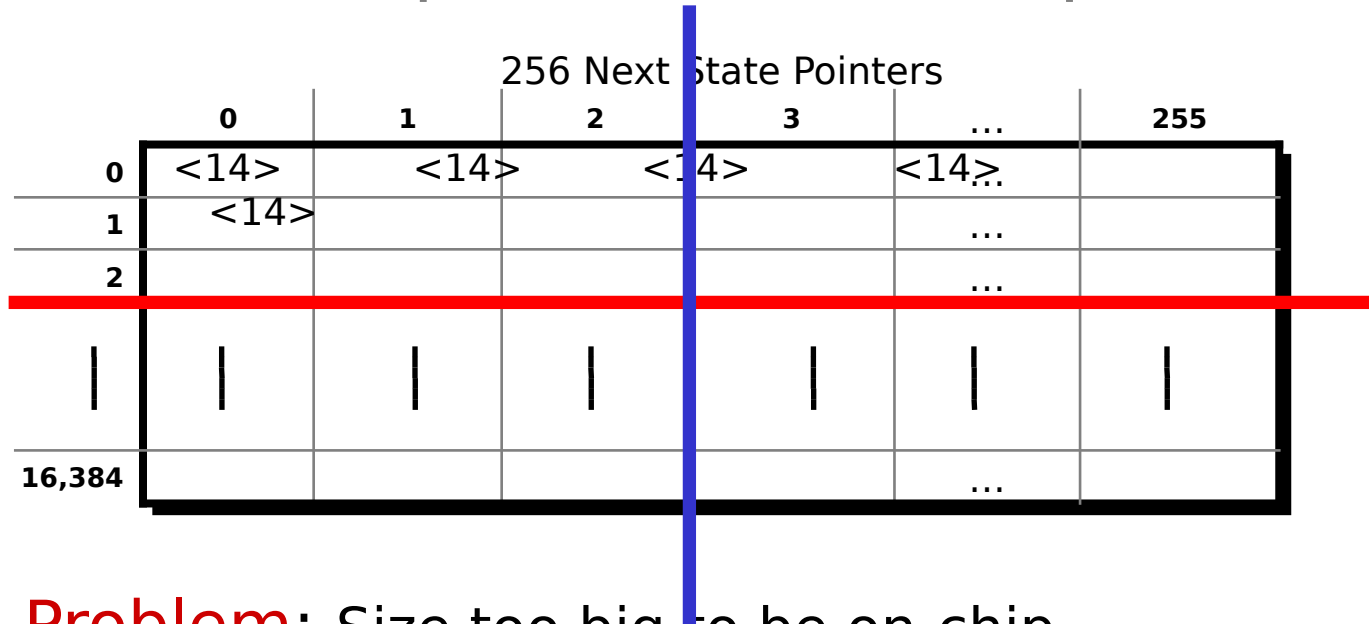
Input
stream:

h x h e r s

Only scan the input stream once.

Linear Time: So what's the problem

How to implement it on chip?



Problem: Size too big to be on-chip

- ~ 10,000 nodes
- 256 out edges per node
- Requires $16,384 \times 256 \times 14 = \sim 10\text{MB}$

Solution: partition into small state machines

- Less strings per machine
- Less out edges per machine

Packet Scan Architecture

String Matching

Bit-Split String Matching Algorithm *many tiny FSM working together*

A Memory Tile Based Architecture

Building a Real System

Is it really correct?

Future Work

An example

$$P_0 = \{ \text{he, she, his, hers} \}$$

Char	0	1	2	3	4	5	6	7
h	0	1	1	0	1	0	0	0
x	0	1	1	1	1	0	0	0
h	0	1	1	0	1	0	0	0
e	0	1	1	0	0	1	0	1

Table 1: Binary Encoding of input stream “hxhe”

An example

$P_0 = \{ \text{he, she, his, hers} \}$

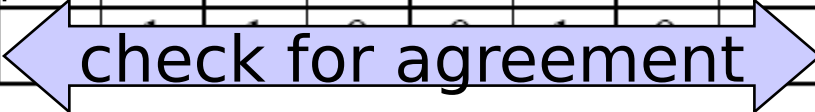
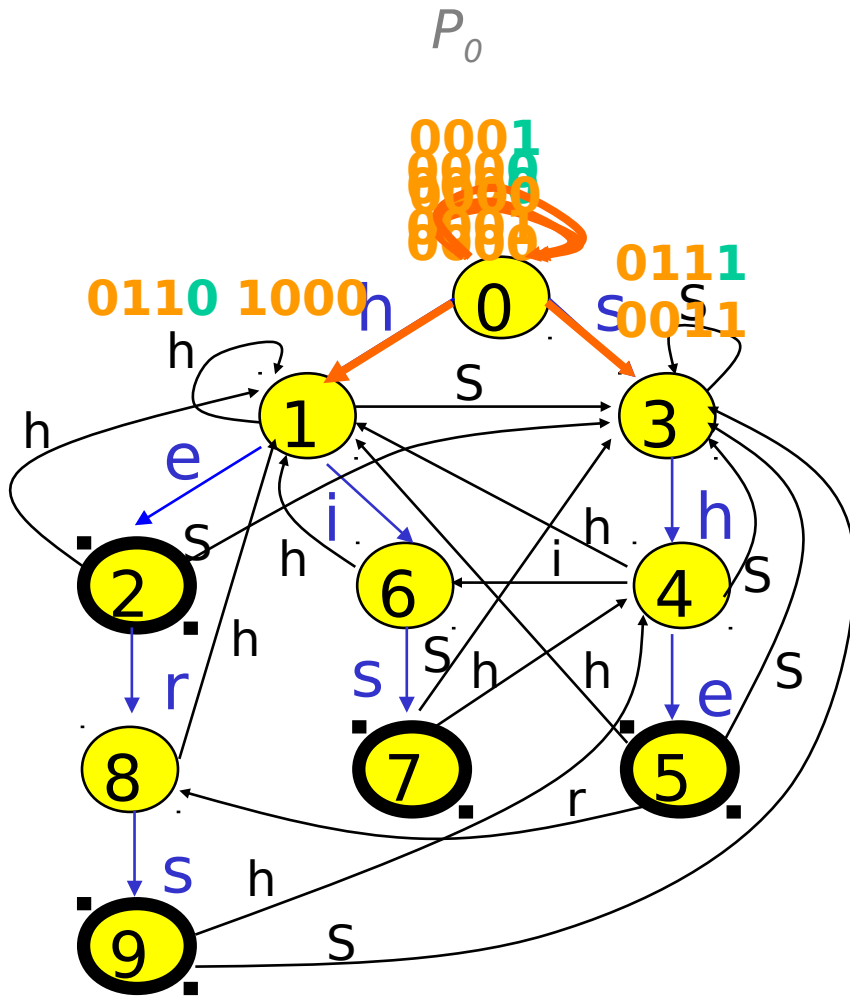
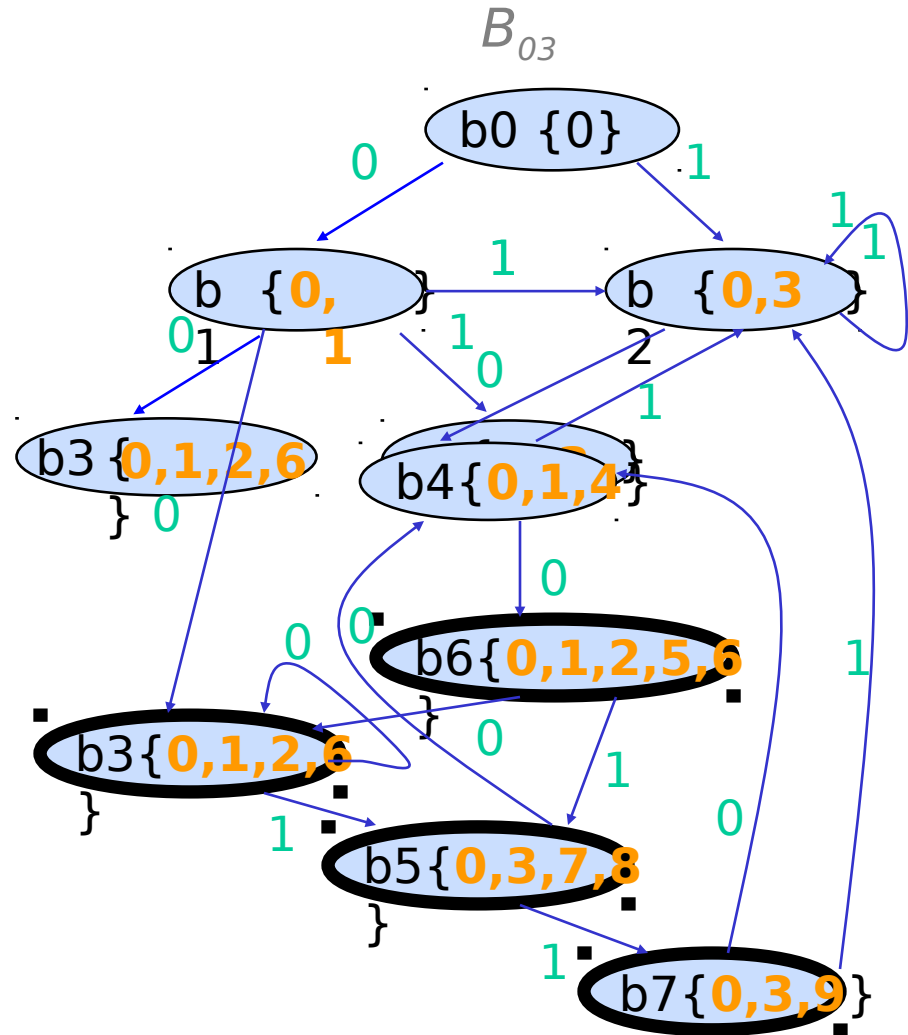
Char	0	1	2	3	4	5	6	7
h	0	1	1	0	1	0	0	0
x	0	1	1	1	1	0	0	0
h	0	1	1	0	1	0	0	0
e								

Table 1: Binary Encoding of input stream "hxhe"

An example of Bit-Split

$$P_0 = \{ \text{he, she, his, hers} \}$$


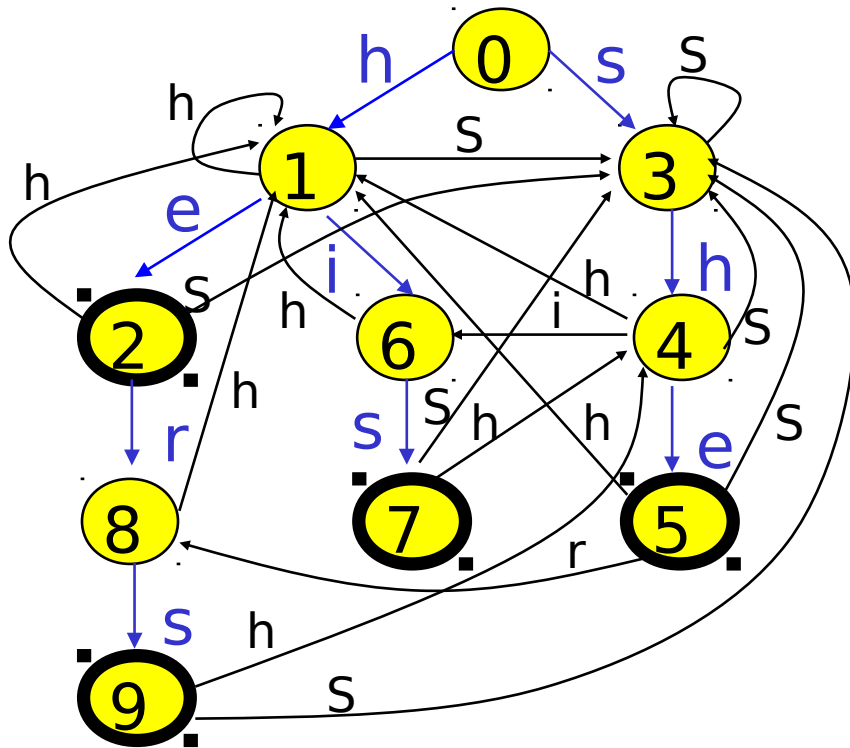
(Edges pointing back to State 0 are not shown).



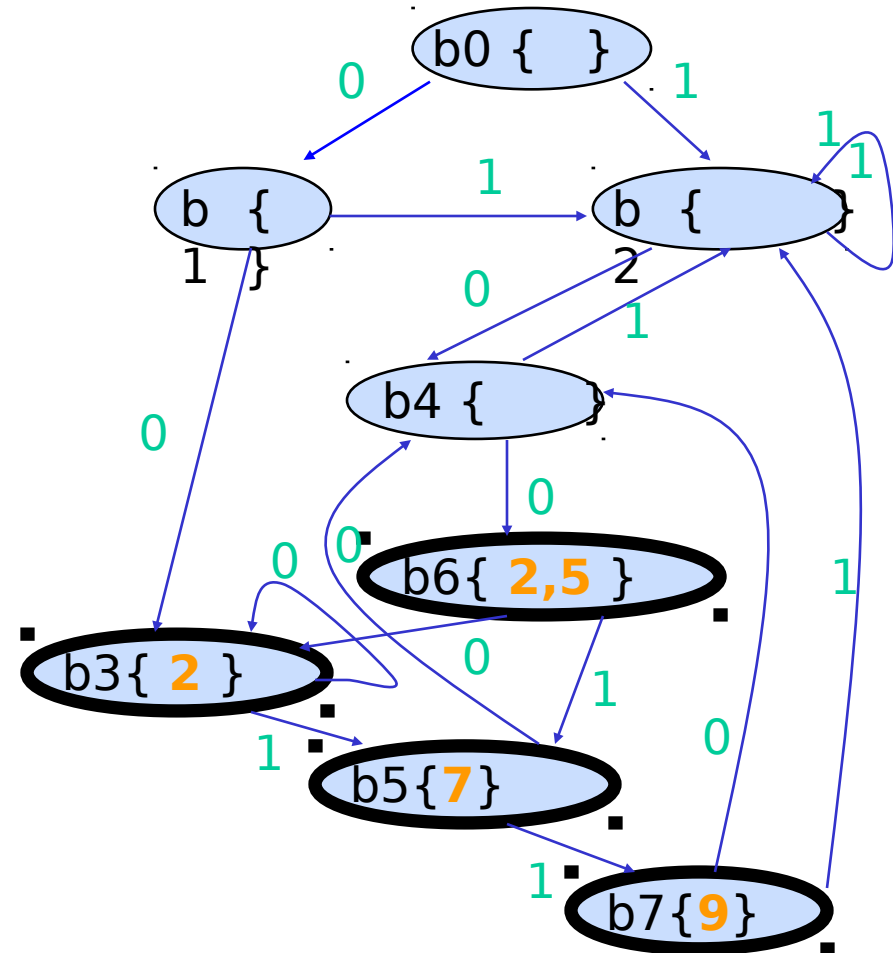
Compact State Set

$$P_0 = \{ \text{he, she, his, hers} \}$$

P_0



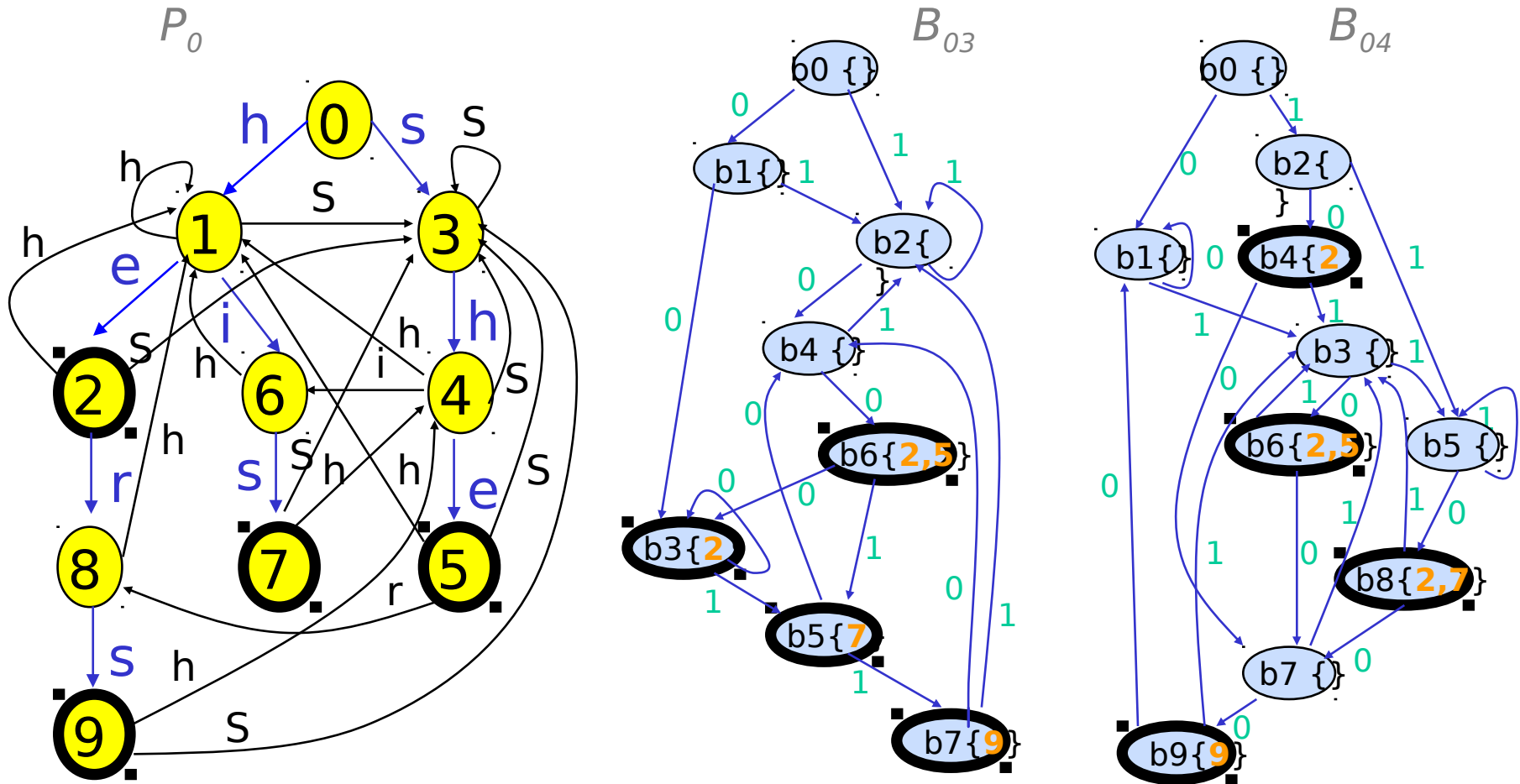
B_{03}



(Edges pointing back to State 0 are not shown).

An example of Bit-Split

$P_0 = \{ \text{he, she, his, hers} \}$



(Edges pointing back to State 0 are not shown).

Nice Properties

The number of states in B_{ij} is rigorously bounded by the number of states in P_i

No exponential blow up in state

Linear construction time

Possible to traverse multiple edges at a time to *multiply* throughput

Matching on the example

hxhe

P_0

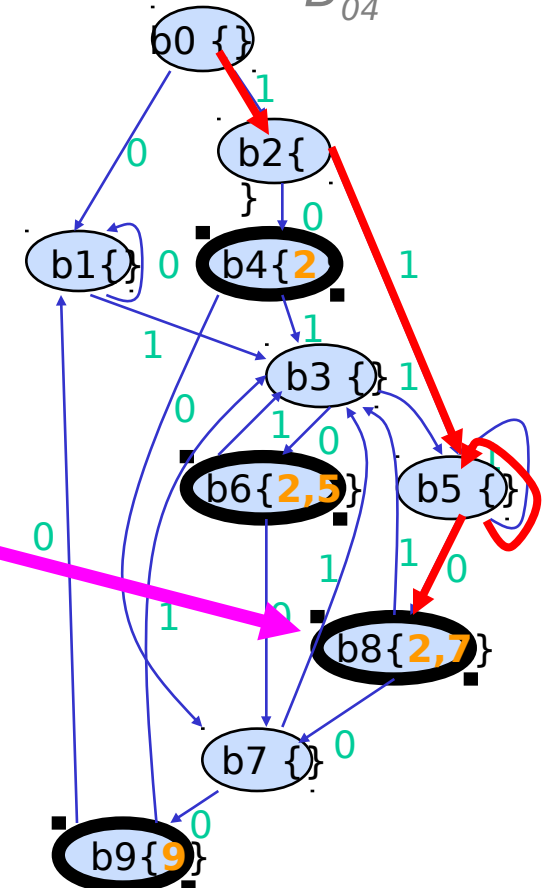
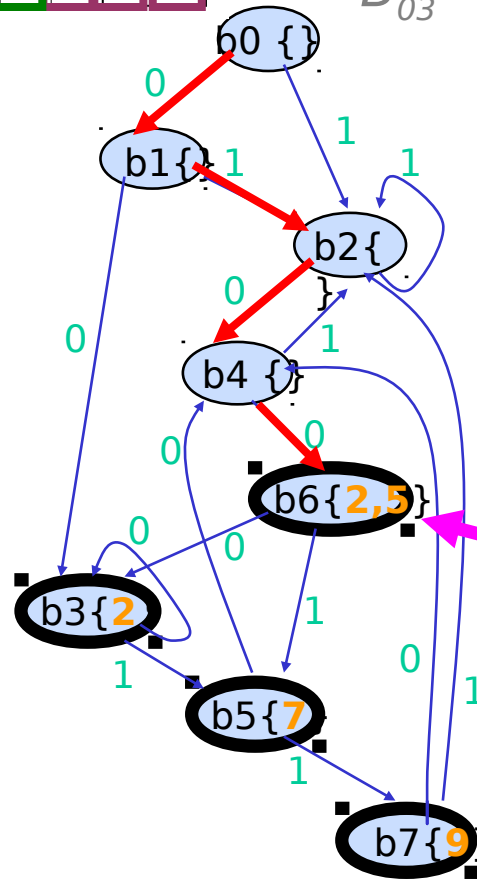
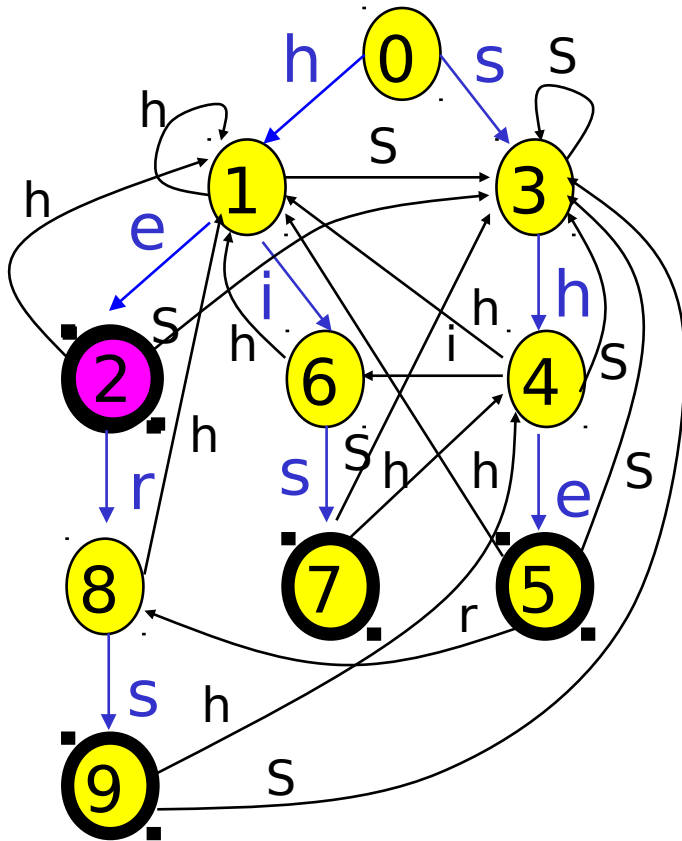
Char	0	1	2	3	4	5	6	7
h	0	1	1	0	1	0	0	0
x	0	1	1	1	1	0	0	0
h	0	1	1	0	1	0	0	0
e	0	1	1	0	0	1	0	1

0100

B_{03}

1110

B_{04}



How do you “combine” the results from the different state machines?

Only if all the state machines agree, is there actually a

Packet Scan Architecture

String Matching

Bit-Split String Matching Algorithm

A Memory Tile Based Architecture

*SRAM tiles
implement FSM*

Building a Real System

Is it really correct?

Future Work

Our Main Idea: Bit-Split

Partition rules (P) into smaller sets (P_0 to P_n)

Build AC state-machine for each subset

For each DFA P_i , rip state-machine apart into 8 tiny state-machines (B_{i0} through B_{i7})

Each of which searches for **1 bit** in the 8 bit encoding of an input character

- Only if *all* the different B machines *agree* can there actually a match

How to Implement

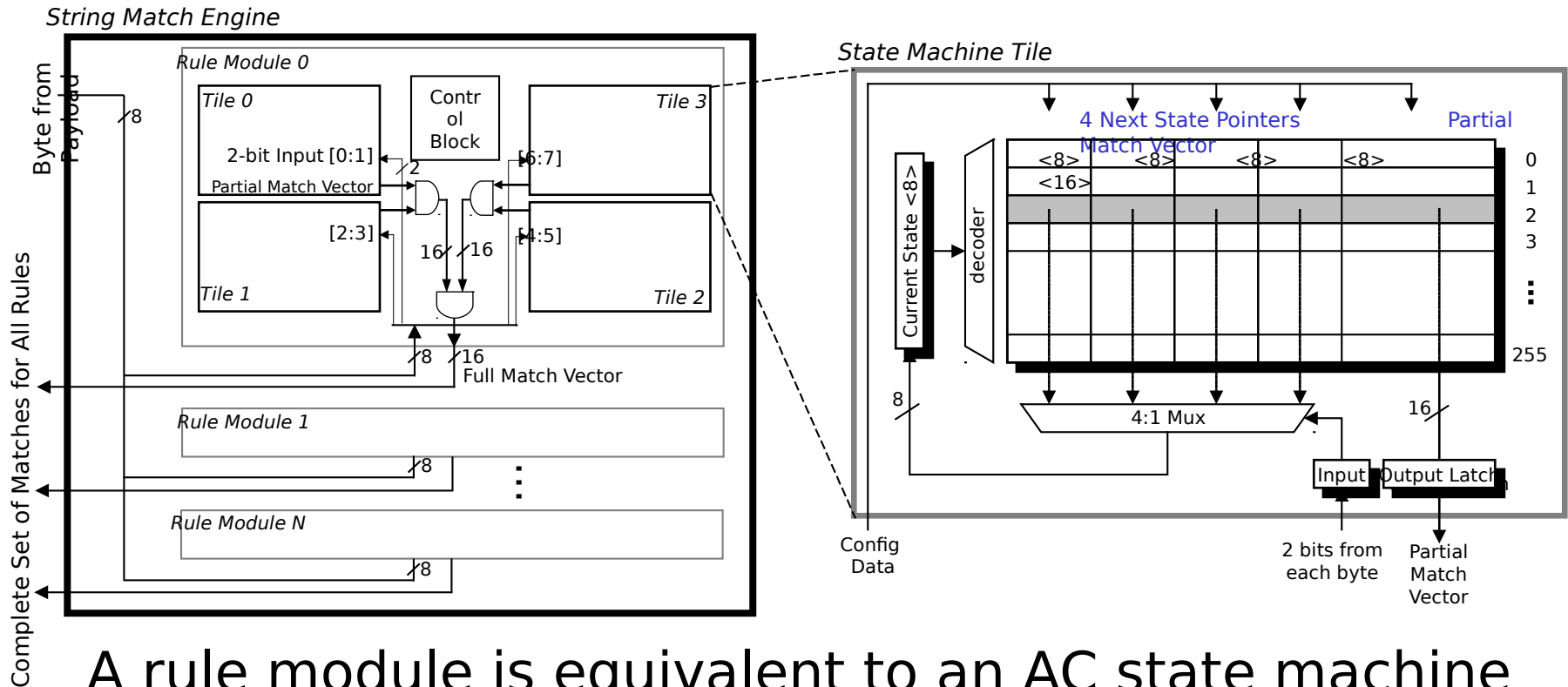
The AC state machine is *equivalent* to the 8 tiny state machines.

The 8 tiny state machines can run *independently*, which means in parallel Intersection done with bit-wise AND.
8 is intuitive but not optimal

How to build a system to implement this algorithm?

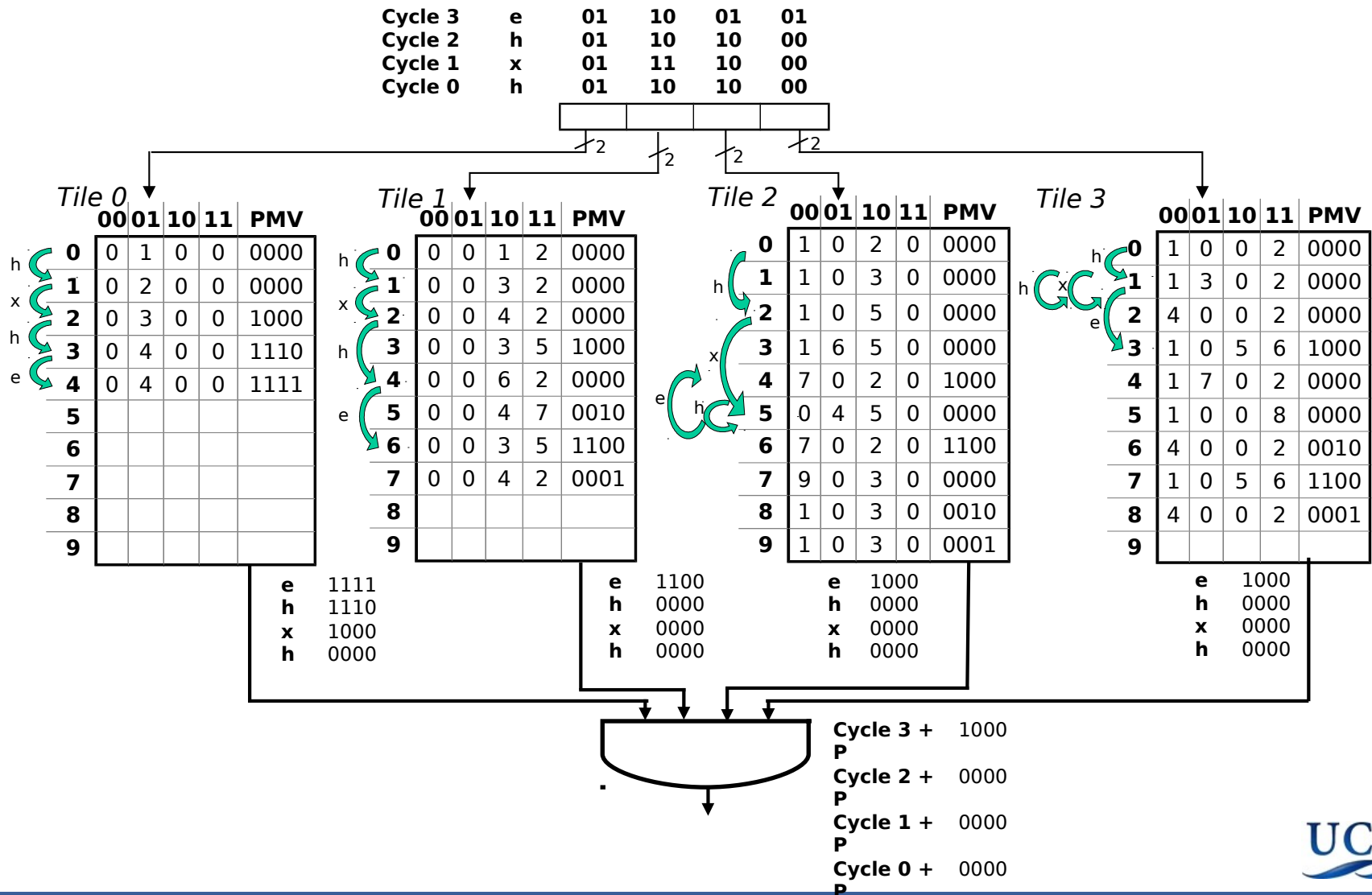
- Our algorithm makes it feasible to be on-chip

A Hardware Implementation

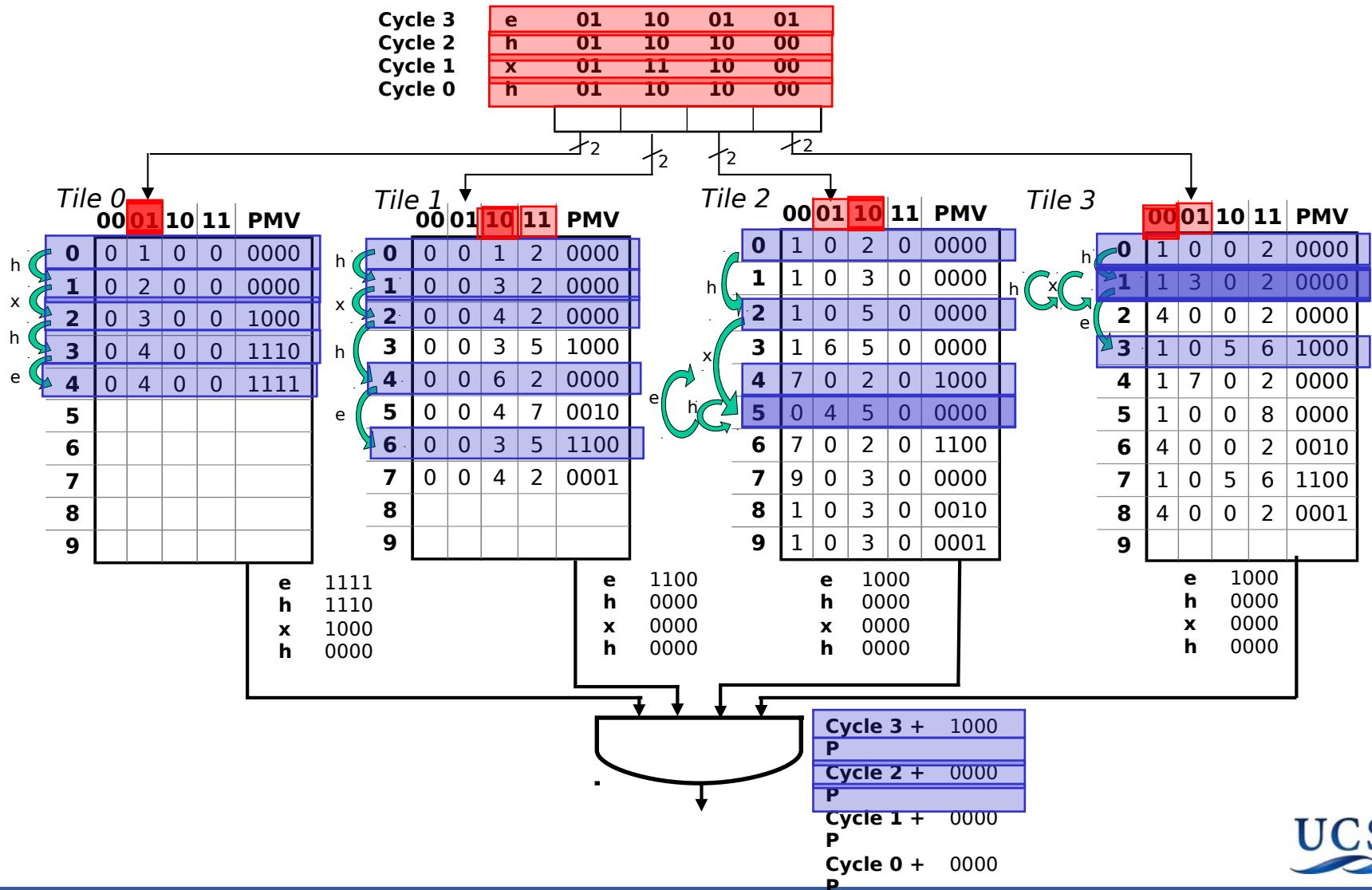


A rule module is equivalent to an AC state machine
Rule modules, tiles are structurally equivalent
All full match vectors are concatenated to indicate
which strings are matched
One tile stores one tiny bit-split state machine

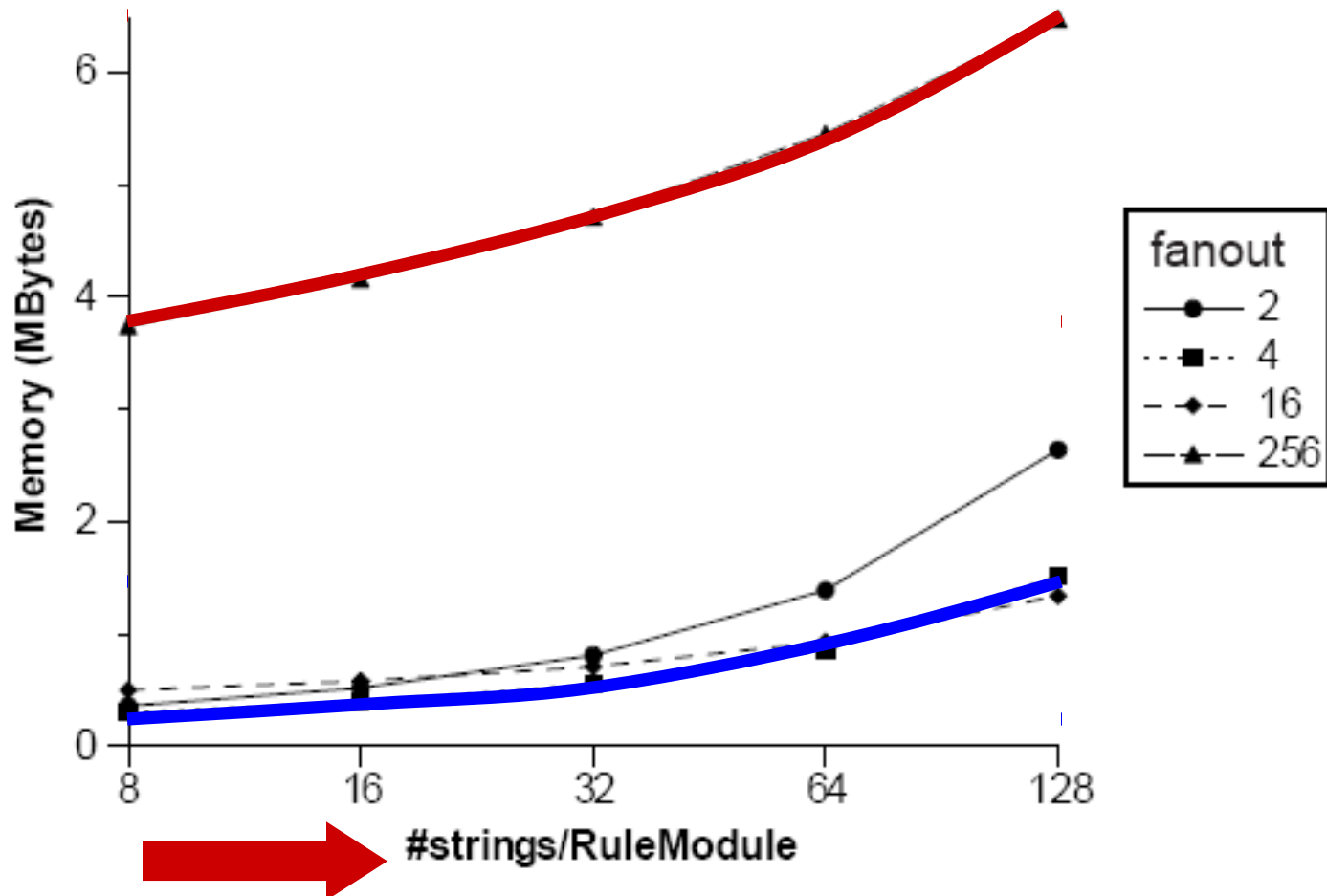
An efficient Implementation



An efficient Implementation

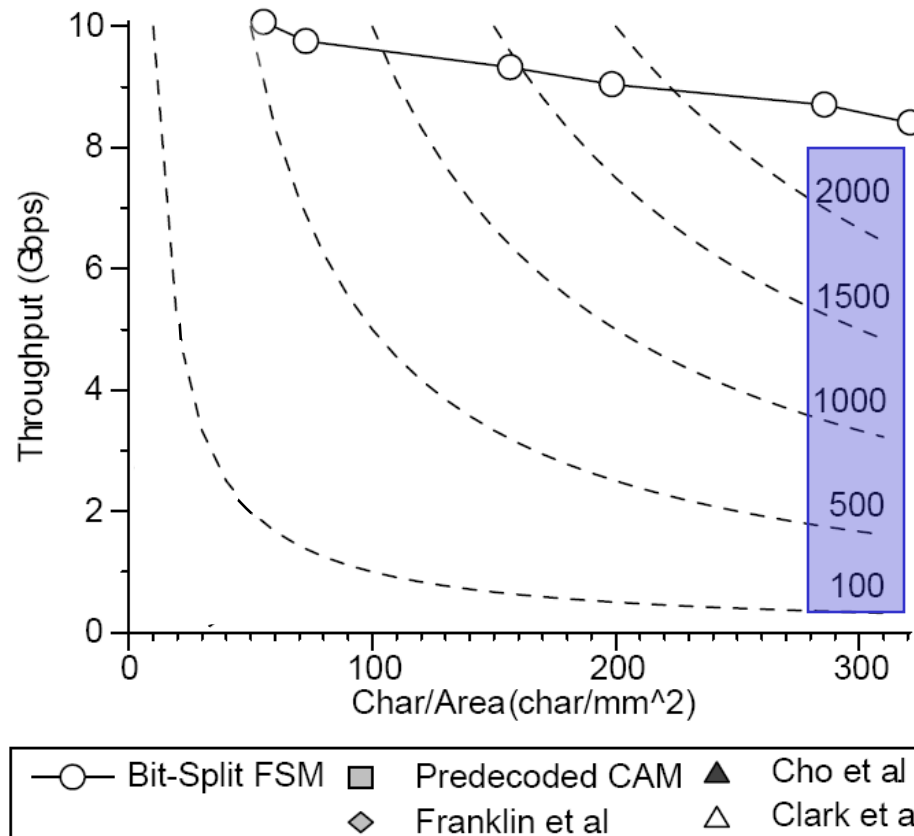


Performance of Hardware



Performance of Hardware

Key Metric: Throughput*Character/Area



Packet Scan Architecture

String Matching

Bit-Split String Matching Algorithm

A Memory Tile Based Architecture

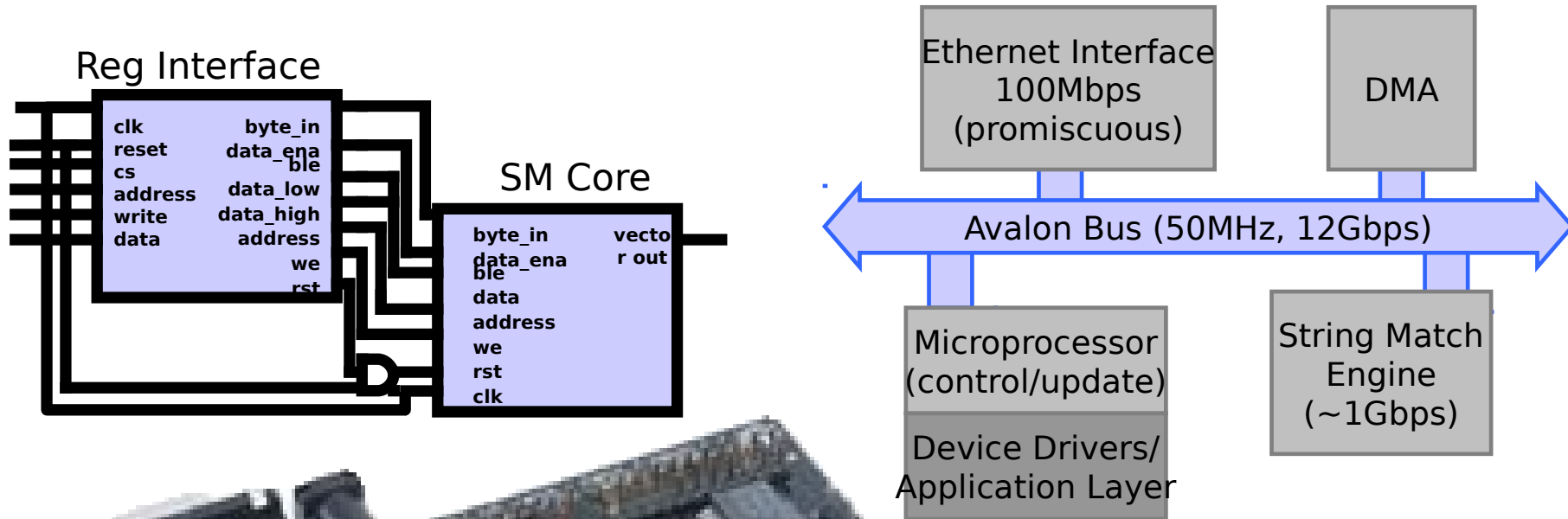
Building a Real System

*Integration and
interfaces (FPGA)*

Is it really correct?

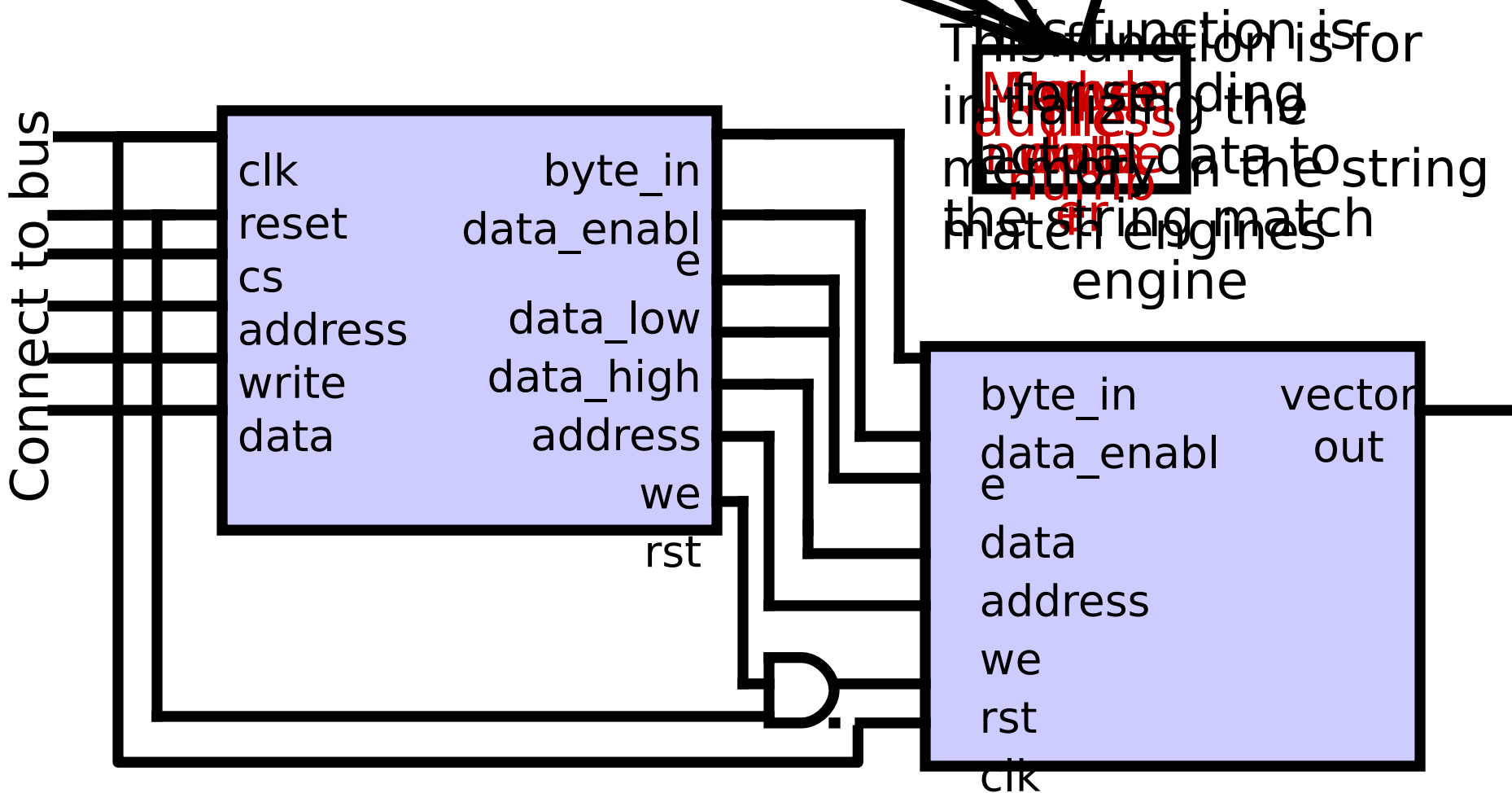
Future Work

Prototype Design



Interface With Avalon Bus

smes_remove_byte_t(BB_based_addr_byte_1 from 0x00000000, 0x00000000)



Packet Scan Architecture

String Matching

Bit-Split String Matching Algorithm

A Memory Tile Based Architecture

Building a Real System

Is it really correct?

*Proofs
(yes)*

Future Work

A Formalization

$$M = (Q, \Sigma, \delta, q_1, F)$$

$Q = \{q_1, q_2, \dots, q_m\}$ is a set of states

q_1 is the start state.

$$\delta : Q \times \Sigma \rightarrow Q$$

$F \subseteq Q$ is the set of final (or accepting) states

$$\Sigma = A \times B \quad \begin{array}{l} A = \{\alpha_1, \alpha_2, \dots, \alpha_r\} \\ B = \{\beta_1, \beta_2, \dots, \beta_s\} \end{array}$$

Cartesian product

$\mathcal{L} = \mathcal{L}(M)$ denote the language accepted by M

Each $w \in \mathcal{L}$ is of the form $w = a_1b_1 a_2b_2 \cdots a_nb_n$

Splits DFA as an NFA

decompose M

$$M_A = (Q, A, \delta_A, q_1, F) \text{ where } \delta_A(q, a) = \bigcup_{i=1}^s \delta(q, a\beta_j)$$

$$M_B = (Q, B, \delta_B, q_1, F) \text{ where } \delta_B(q, b) = \bigcup_{i=1}^r \delta(q, \alpha_i b)$$

both M_A and M_B are nondeterministic finite automata (NFA)

languages $\mathcal{L}_A = \mathcal{L}(M_A)$ and $\mathcal{L}_B = \mathcal{L}(M_B)$ are homomorphic images of \mathcal{L}

Correctness stems from RL subset

$$\text{Alt}(\mathcal{L}_A, \mathcal{L}_B) = \{a_1b_1 a_2b_2 \cdots a_nb_n \mid n \geq 0, a_1a_2 \cdots a_n \in \mathcal{L}_A, b_1b_2 \cdots b_n \in \mathcal{L}_B\}.$$

$$\mathcal{L} \subseteq \text{Alt}(\mathcal{L}_A, \mathcal{L}_B)$$

Suppose \mathcal{L} has the property that

$$a_1y_1 a_2y_2 \cdots a_ny_n, x_1b_1 x_2b_2 \cdots x_nb_n \in \mathcal{L} \text{ implies } a_1b_1 a_2b_2 \cdots a_nb_n \in \mathcal{L}$$

for every $n \geq 0$, $a_i, x_i \in A$, $b_i, y_i \in B$ for $i = 1, 2, \dots, n$. Then $\mathcal{L} = \text{Alt}(\mathcal{L}_A, \mathcal{L}_B)$.

The above property is sufficient, is it necessary?

Exploiting fixed wildcards is possible, what about more general patterns?

Packet Scan Architecture

String Matching

Bit-Split String Matching Algorithm

A Memory Tile Based Architecture

Building a Real System

Is it really correct?

Future Work

*Extensions
and Applications*

Primitives for Security

Packet Address List Lookup

Packet Address Range Query

Packet Classification

String Finding

Regular Expression Finding

Statefull Flow Monitors

Packet Ordering

Related Work

Software based

- Good for $\sim 100\text{Mb/s}$, common case

FPGA-based

- Many schemes map rules down to a specialized circuit
 - Near optimal utilization of hardware resources
- Implementing state machines on block-RAMs [Cho and Mangione-Smith]
- Concurrent to our work: mapping state machines to on-chip SRAM [Aldwairi et. al.]
- Bloom filters [Dharmapurikar et al.]
 - Excellent filter in the common case

TCAM-based

- Require all patterns to be shorter or equal to TCAM width
- Cutting long patterns: 2Gbps with 295KB TCAM [Yu et. al.]

Conclusions

New Tile-based Architecture

- 0.4MB and 10Gbps for Snort rule set (>10,000 characters)
- Possible to be used for other applications, e.g. IP lookups, packet classification.

New Bit-split Algorithm:

- *General* purpose enough for many other applications, e.g. spam detection, peephole optimization, IP lookups, packet classification, etc.
- Feasible to be implemented on other tile-based architecture.

Thanks

Lin Tan

Brett Brotherton

Prof. Ryan Kastner

Prof. Ömer Egecioglu

Shreyas Prasad, Shashi Mysore, Bitu
Mazloom, Ted Huffmire, Banit Argawal

All done.